Introducció al programari distribuït

Benet Campderrich Falgueras

P00/05007/00305

UOC
www.uoc.edu

Índex

Introducció

Tots sabem que cada vegada són més rars els ordinadors que no formen part d'una xarxa (més aviat, hauríem de dir de la xarxa mundial). Per bé que cadascun dels ordinadors d'una xarxa podria tenir un programari independent que s'executés solament de manera local, en moltes aplicacions és avantatjós que col·laborin diversos ordinadors, i les modalitats, conceptes i eines bàsiques per a aquesta col·laboració és el que es veu –de manera introductòria, certament– en aquest mòdul didàctic.

Objectius

L'objectiu general d'aquest mòdul és donar als estudiants una presentació general de la tecnologia actual de programari distribuït, que els serveixi alhora de base i d'estímul per a estudis més avançats en aquest camp que té un desenvolupament molt ràpid i que sembla que adquirirà cada vegada més importància durant uns quants anys. Aquest objectiu general es descompon en els següents:

1. Presentar les idees fonamentals dels entorns distribuïts i dels sistemes oberts, que són la base damunt la qual es basteix el programari distribuït.

2. Descriure els entorns distribuïts més importants en ordre de complexitat creixent.

3. Introduir CORBA, que es pot considerar actualment l'arquitectura i infraestructura tecnològica més general per a programari orientat a objectes distribuïts.

4. Donar les idees bàsiques sobre el que es pot aconseguir amb l'RMI, que és la part de Java que suporta la col·laboració entre objectes distribuïts.

5. Presentar la tecnologia de documents compostos –distribuïts o no– en general, i la tecnologia OLE/COM/DCOM en particular.

6. Presentar, de manera introductòria, la problemàtica específica del disseny de la distribució del programari orientat a objectes.

1. Entorns distribuïts i entorns oberts

Parlem d'*entorn distribuït* quan el programari d'una aplicació s'executa repartit entre diversos ordinadors d'una xarxa; llavors, també diem que el programari en qüestió és distribuït.

La utilització d'entorns distribuïts és relativament recent, i s'hi ha arribat mitjançant la substitució de plataformes de maquinari i programari bàsic més antigues. Simplificant, es pot dir que la raó decisiva per a l'evolució cap als entorns distribuïts és que la mateixa capacitat de procés surt molt més barata* si s'aconsegueix amb PC que amb *mainframes* o màquines Unix.

* Aproximadament dos ordres de magnitud més barata.

L'evolució de les plataformes

D'una banda, els terminals de pantalla van arraconar les targetes perforades i després les van substituir els microordinadors que al principi les emulaven; d'una altra, els ordinadors mitjans (miniordinadors) van reemplaçar, en part, els *mainframes*, i després les xarxes locals i els sistemes client/servidor han ocupat el lloc dels sistemes amb miniordinadors i *mainframes* que quedaven.

1.1. Objectius dels entorns distribuïts

Els entorns distribuïts van ser creats per a assolir els objectius següents:

1) Portabilitat d'aplicacions i de serveis del sistema, que vol dir que totes dues es puguin executar indistintament en diversos ordinadors.

2) Interoperabilitat, que vol dir que els seus ordinadors i aplicacions diferents siguin capaços de comunicar-se per mitjà de comandes i formats de dades que tots aquests comprenguin.

3) Integració, que significa que els intercanvis d'informació es puguin fer sense necessitat d'intervencions externes*, i també que el funcionament del programari i la presentació de la informació tinguin una certa uniformitat.

* Per exemple, compartint directament la informació.

4) Transparència, en el sentit que els usuaris puguin llegir dades d'un ordinador sense saber on és (transparència en la ubicació de les dades), i els processos es puguin executar indistintament en diversos ordinadors sense que els usuaris sàpiguen a quin s'executen (transparència en l'execució).

5) Facilitat de creixement del sistema, que té dos vessants: que es pugui afegir amb facilitat maquinari o programari i que els components d'aquests es puguin substituir sense gaire trasbals per altres de més avançats o més potents.

6) Compartició, en el sentit que les aplicacions, recursos i serveis puguin ser compartits sense barreres tècniques (una altra cosa són les restriccions d'accés intencionades).

7) Finalment, també és important la **seguretat**, que consisteix en el fet que les dades d'un usuari estiguin protegides, quant a consulta i quant a actualització, dels altres usuaris i dels agents externs, i les comunicacions no es puguin espiar.

1.2. Importància de les normes en els entorns distribuïts

Perquè una arquitectura distribuïda es pugui considerar ben dissenyada cal que tingui potencialment una vida molt més llarga que els productes amb què està implementada.

Per a aconseguir-ho, cal que l'arquitectura es basi en normes àmpliament acceptades, cosa que constitueix una certa garantia que durant força anys es trobaran al mercat productes que les compleixin, i que per tant es puguin fer servir per a renovar i ampliar el sistema i augmentar-ne la capacitat. Cal que l'estandardització s'apliqui en els camps següents: ordinadors, plataformes de suport de la interfície amb l'usuari, sistemes operatius, protocols de comunicacions i sistemes operatius de xarxa.

1.3. Concepte de sistema obert

Un sistema obert és un sistema distribuït en el qual s'intenta d'aconseguir, almenys parcialment, els objectius dels sistemes distribuïts i fer que les interfícies entre els seus components respectin un conjunt ampli i complet de normes sobre comunicacions, programació, presentació i interfícies entre aplicacions i serveis del sistema acceptades internacionalment.

Les interfícies en els sistemes oberts

En un sistema obert no cal, per exemple, que tots els sistemes operatius siguin de tipus Unix, ni cal prescindir dels sistemes operatius propis d'una marca, sempre que tots els sistemes operatius utilitzats compleixin les normes que facin possible el grau necessari de comunicació mitjançant unes interfícies adequades (dit d'una altra manera: que un sistema sigui obert o no, no és cosa dels productes en si, sinó de les seves interfícies).

Quan parlem de normes en aquest context volem dir el següent:

- o bé –i preferentment– normes àmpliament utilitzades establertes per organitzacions independents,

- o bé normes que són inicialment propietat d'un fabricant, però que han esdevingut estàndards *de facto*.

Per tant, per tal d'aconseguir un sistema obert caldria evitar el següent:

- els productes molt utilitzats però tancats, en el sentit que difícilment poden compartir dades amb altres productes,

- i aquelles normes que, per les raons que siguin, hagin estat objecte d'escasses implementacions.

1.3.1. Beneficiaris dels sistemes oberts

Els sistemes oberts beneficien els grups següents:

- Els **fabricants de maquinari i sistemes operatius**: en la mesura que alguns aspectes del producte estan fixats per normes i ja no cal estudiar-ne alternatives, es pot dedicar més esforç a millorar-ne les arquitectures i la funcionalitat.

- Els **fabricants independents de programari** es beneficien d'un mercat més ampli a conseqüència de la portabilitat, fet que evita la necessitat de versions específiques per a cada plataforma.

- Les **empreses usuàries** es beneficien de poder connectar sense problemes productes de fabricants diferents i de trobar més productes en el mercat (ja que molts productes passen a ser utilitzables en moltes plataformes) i amb preus més baixos per l'augment de la competència.

2. Entorns client/servidor clàssics

Actualment, l'entorn distribuït que es fa servir més és l'anomenat *client/servidor*.

La idea bàsica de l'**arquitectura client/servidor** és que un programa, el **servidor**, gestiona un recurs compartit concret i fa determinades funcions només quan les demana un altre, el **client**, que és el que interactua amb l'usuari.

Exemple d'un entorn client/servidor

Un exemple senzill d'entorn client/servidor seria tenir un sistema de gestió de base de dades engegat en un servidor i que els programes que s'executen als ordinadors clients puguin fer instruccions d'SQL per a accedir a la base de dades en qüestió.

Normalment, aquests dos programes, el servidor i el client, estan en ordinadors diferents.

Els requeriments dels ordinadors clients quant a velocitat, memòria i capacitat de disc són molt diferents dels dels servidors; uns i altres poden ser ordinadors de model i marca diferents i, a més, sovint fan servir un sistema operatiu diferent.

2.1. Avantatges i inconvenients de l'arquitectura client/servidor

Els **avantatges de l'arquitectura client/servidor** són els següents:

a) Permet que la informació es processi a prop d'on s'ha generat.

b) Com que les funcions del programari queden repartides entre diverses màquines, és possible de fer servir PC o estacions de treball per als clients, i màquines Unix (per exemple) per als servidors, totes aquestes d'un cost molt menor que els *mainframes*.

c) El creixement del maquinari pot ser gradual:

- es pot augmentar gradualment el nombre de clients sense que calgui canviar cada vegada el servidor;
- es pot substituir el servidor sense que els clients es vegin afectats;
- es pot augmentar la capacitat d'un client sense haver de canviar ni el servidor ni els altres clients;
- en algunes modalitats, es poden afegir servidors sense haver de redissenyar l'arquitectura en el seu conjunt.

d) Facilita l'ús d'interfícies gràfiques d'usuari i aplicacions multimèdia; la raó és que aquestes funcions necessiten el tractament d'un volum important d'informació, que si s'hagués de fer de manera centralitzada necessitaria un *host* de gran capacitat i a més unes trameses d'informació molt considerables mitjançant les línies.

Molts d'aquests avantatges signifiquen simplement que s'aconsegueixen, en grau més alt o més baix, els objectius dels entorns distribuïts.

Vegeu els objectius dels entorns distribuïts en el subapartat 1.1 d'aquest mòdul didàctic

Els **inconvenients de l'arquitectura client/servidor** són els següents:

a) El servidor pot ser un coll d'ampolla.

b) El programari distribuït és més complex que el no distribuït, i també és més difícil de provar i depurar errors; també l'administració i la problemàtica de seguretat són bastant més complexes.

c) Qualsevol sistema distribuït tendeix a fallar més sovint que un sistema centralitzat, ja que té més components que poden fallar independentment.

2.2. Arquitectures client/servidor de dues capes

Ara veurem diferents entorns d'arquitectures client/servidor de dues capes:

1) Entorns client/servidor de primera generació: és típic de les xarxes d'àrea local (LAN); els clients són PC o estacions de treball, en els quals s'executen les aplicacions, i els servidors, al principi, només fan funcions generals i de baix nivell com ara gestionar fitxers o impressores compartides. Més endavant, es comencen a passar part de les aplicacions a un servidor addicional, si bé els clients continuen iniciant i controlant els processos en part.

2) Entorns d'igual a igual*: un ordinador fa de client per a d'altres màquines i de servidor per a aquestes mateixes o unes altres, fins i tot per a si mateix.

*** En anglès, *peer-to-peer*.**

3) Entorns client/servidor de segona generació: hi ha diversos servidors especialitzats en funcions diferents que poden demanar els clients: gestió de bases de dades, aplicacions, etc.; els clients poden ser mòbils.

2.2.1. Limitacions de l'arquitectura client/servidor de dues capes

Aquest tipus d'arquitectures presenten les limitacions següents:

- Limitacions al creixement del nombre de clients, ja que tots es connecten directament al servidor.

- Dificultat de mantenir el programari dels clients, ja que qualsevol canvi s'ha de fer en tots alhora.

2.3. Arquitectures de més de dues capes

2.3.1. Arquitectura de tres capes basada en les plataformes

En el cas més general d'arquitectura de tres capes basada en plataforma, l'estructura es la següent:

1) Primera capa: el **servidor**
2) Segona capa: els **agents**
3) Tercera capa: els **clients**

Els agents poden fer funcions com ara aquestes:

Exemple de traducció d'aplicacions

Una traducció d'aplicació consisteix, per exemple, a adaptar a un entorn client/servidor una aplicació antiga damunt el servidor.

- Traducció d'aplicacions.

- Control de la càrrega del servidor.

- Serveis d'agent intel·ligents, com descompondre una petició de servei per part d'un client en diverses peticions a un o diversos servidors; si hi hagués diversos servidors, això representaria una certa transició cap al *middleware* (programari intermedi).

Vegeu el concepte de *middleware* en el subapartat 3.1 d'aquest mòdul didàctic.

Aquest model es pot expandir horitzontalment a tots els nivells amb una certa facilitat. L'inconvenient principal és la forta dependència de les plataformes utilitzades.

2.3.2. Arquitectura de tres capes basada en les funcions

Aquest tipus d'arquitectura presenta les tres capes següents:

1) Clients, en els quals hi ha la lògica de tractament de la interfície amb l'usuari i part de la lògica d'aplicació.

2) Servidors d'aplicació, en els quals resideix la major part de la lògica d'aplicació i de manipulació de dades.

3) Servidors de dades, en els quals hi ha els gestors de bases de dades.

Ara convé que feu els exercicis d'autoavaluació de l'1 al 4 d'aquest mòdul didàctic.

3. Entorns amb *middleware*: CORBA

3.1. Concepte de *middleware*

Quan la dimensió de la xarxa creix encara més i incorpora una multiplicitat d'aplicacions, plataformes i xarxes, cal un component que gestioni la comunicació entre tot això; aquest component va col·locat entre els clients i els servidors, i per aquest motiu s'anomena **programari intermedi** o ***middleware***.

El programari intermedi ha de suportar diferents protocols i interfícies de comunicacions i d'accés a les dades i permetre connexions dinàmiques entre clients i servidors.

3.2. CORBA

El programari distribuït es pot desenvolupar amb tecnologia orientada a objectes i amb tecnologia clàssica. Tanmateix, la manera com funciona el programari orientat a objectes, per mitjà de missatges entre objectes que demanen l'execució d'operacions, es presta molt bé a la distribució, ja que un missatge a un objecte que està en un altre ordinador pot ser igual, quant a la forma, que un missatge a un objecte que està en el mateix ordinador, si hi ha la infraestructura necessària per a fer arribar cada missatge al seu destinatari sigui on sigui.

Dit d'una altra manera, qualsevol programari orientat a objectes funciona per crides a operacions entre un objecte que fa de client i un altre que fa de servidor; observeu que diem "fa" i no pas "és", perquè el mateix objecte pot fer de client en una crida i de servidor en una altra.

La ***Common Object Request Broker Architecture*** (**CORBA**) és una arquitectura per a sistemes d'objectes distribuïts desenvolupada per l'OMG*. Comprèn les especificacions d'un programari intermedi orientat a objectes dissenyat per a oferir portabilitat i interoperabilitat dins una xarxa de sistemes heterogenis.

*** L'OMG és la mateixa organització que va desenvolupar UML.**

Vegeu l'OMG en el subapartat 5.1 del mòdul "Introducció a l'enginyeria del programari OO" d'aquesta assignatura.

3.2.1. Terminologia bàsica

A continuació veurem alguns conceptes importants per a la descripció de CORBA:

1) Interfície

Sabem que en el programari orientat a objectes, i com a conseqüència de la seva encapsulació, els objectes es comuniquen per mitjà de missatges en els quals es demanen operacions i valors d'atributs –tots dos públics o, a tot estirar, protegits– i que, consegüentment, es distingeix entre una classe i la interfície o interfícies que implementa.

Ara bé, en el cas de programari no distribuït, l'objecte client i l'objecte servidor estan en el mateix procés, i, per tant, la interfície i la classe que la implementa també; en el cas de programari distribuït, el client i el servidor poden estar en processos diferents i, fins i tot, en màquines diferents, i per tant, en el procés del client no hi ha la classe del servidor, sinó només la interfície d'aquesta. És clar que hi continua havent una classe que implementa la interfície, però aquesta classe no és visible des del client, sinó només la seva interfície, com hem dit. Per tant, el programari intermedi –i per tant CORBA– només considera les interfícies, ja que el concepte de *classe* no es necessita per a la comunicació entre objectes clients i objectes servidors; i tampoc no cal per a la implementació, ja que són els objectes, i no les classes, els que implementen les interfícies, perquè qualsevol operació es demana a un objecte concret.

Les classes en l'entorn CORBA

Dins el programa que es fa servir per a implementar un objecte determinat sí que es poden definir classes, però com que són únicament d'àmbit local CORBA no les "veu".

A més de les interfícies dels objectes de l'aplicació, que són pròpies de cada aplicació, hi ha interfícies estàndard, la definició de les quals forma part de les especificacions de CORBA, que es fan servir en les comunicacions entre l'aplicació i els components de CORBA i també entre aquests.

Les interfícies en CORBA i en UML

El concepte d'*interfície* en CORBA és molt semblant al d'UML (com escau en haver estat tots dos especificats per la mateixa organització, però, tanmateix, CORBA no fa referències a UML).

Com a diferències una mica significatives trobem que les interfícies d'UML no inclouen atributs i les de CORBA sí (però aquesta diferència no té cap repercussió efectiva, ja que un atribut pot ser substituït per dues operacions, una que li posa el valor i una altra que el llegeix), i que en CORBA no hi ha operacions ni atributs de classe.

Hi ha herència entre interfícies; les anomenades **interfícies derivades** hereten de les respectives interfícies bàsiques; hi pot haver interfícies abstractes, semblants a les classes abstractes en el sentit que no tenen atributs perquè no hi pot haver objectes que les satisfacin directament, sinó només mitjançant interfícies derivades d'aquestes.

Herència múltiple entre les interfícies

Hi pot haver herència múltiple –si més no, n'hi ha entre les interfícies estàndard esmentades– però és estrictament additiva (és a dir, no hi ha coincidència de noms entre les operacions o atributs que s'hereten d'interfícies bàsiques diferents). En les interfícies estàndard es donen alguns casos de polimorfisme, en els quals operacions del mateix nom es comporten de manera diferent en el cas de la interfície derivada que en el de la bàsica.

2) Objecte

Dins aquest model, un objecte és una entitat que proporciona serveis (operacions) a les entitats que els demanin (els clients). Els objectes tenen identitat, interfície i implementació.

3) Referència a un objecte

Una referència a objecte és quelcom que identifica el mateix objecte cada vegada que es fa servir; un objecte pot tenir diverses referències diferents. El format i valor de les referències a objectes poden dependre de l'ORB.

La unicitat de les referències

No és obligat que no hi pugui haver mai dos objectes amb la mateixa referència, però sí que no pugui haver-n'hi dins d'un àmbit espacial i temporal prou ampli perquè els conflictes siguin pràcticament impossibles.

4) Tipus d'objecte

Un tipus d'objecte és un tipus els membres del qual són referències a objectes. Un tipus d'objecte correspon a una interfície, en el sentit que els objectes corresponents a les referències d'un tipus satisfan la interfície corresponent.

5) Requesta (*request*)

La requesta és el missatge d'un objecte client a un objecte servidor per a demanar l'execució d'un mètode d'aquest darrer objecte. Una requesta comprèn la identificació de l'objecte que l'ha de fer, una operació, zero o més arguments que corresponen als paràmetres de l'operació definits en la interfície utilitzada, i eventualment un context. Els paràmetres de les operacions són essencialment com els de les operacions en UML; s'identifiquen per posició i cadascun té un mode (*in*, *out* o *inout*) i el tipus. El resultat, si n'hi ha, és un paràmetre *out* especial.

La invocació d'una requesta pot ser de les dues formes següents:

- **Estàtica**, i llavors la interfície que fa servir queda determinada en temps de compilació.
- **Dinàmica**, i aleshores la interfície no es determina fins al temps d'execució (si bé la interfície s'ha d'haver compilat abans)

6) Operació i mètode

Dins una requesta s'invoca una operació d'una interfície, però el que s'executa realment és un mètode que fa part de la implementació de l'objecte servidor al qual està adreçada l'operació.

7) Object Request Broker (ORB)

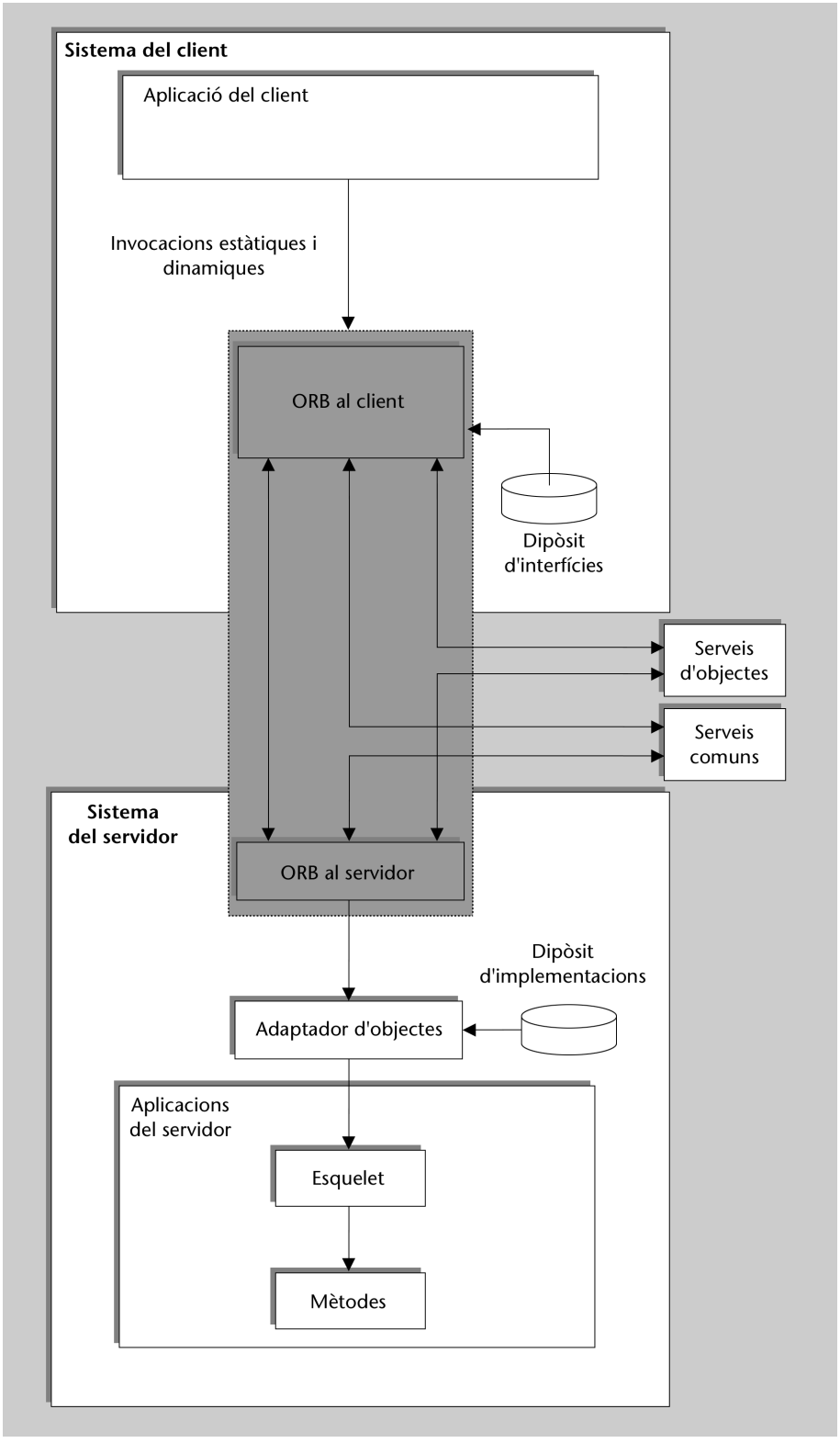
ORB és la sigla d'*Object Request Broker*.

Atès que CORBA es una arquitectura d'objectes distribuïda amb programari intermedi, és lògic que el seu nucli sigui un component responsable de la distribució de missatges entre objectes; aquest component s'anomena *Object Request Broker* (ORB); l'ORB és responsable de trobar l'objecte servidor dins la xarxa, de preparar-lo per a rebre la requesta, de transmetre'l i de fer arribar al client les dades retornades com a resultat de la requesta o resposta a aquesta.

L'activitat de l'ORB fa que l'objecte client només hagi de conèixer la interfície i el valor d'una referència que l'identifiqui entre els altres per a invocar una requesta damunt un objecte servidor, però no el lloc ni el llenguatge en què està programat.

3.2.2. Arquitectura de CORBA

La figura següent representa l'arquitectura de CORBA:



3.2.3. Descripció dels components de l'arquitectura

A continuació, farem una descripció dels components de l'arquitectura que estem considerant:

- **Aplicacions del client**: per mitjà de CORBA els clients fan requestes en què invoquen operacions damunt d'objectes dels servidors mitjançant invocació estàtica o dinàmica. Amb totes dues formes d'invocació es poden fer les mateixes requestes, i el servidor no pot conèixer si s'ha fet servir l'una o l'altra.

- ***Stub***: n'hi ha d'haver un per a cada interfície utilitzada; estableix la correspondència entre les operacions definides a la interfície i les rutines dependents del llenguatge de programació que crida l'aplicació del client quan fa una requesta. Es genera a partir de la interfície descrita en l'IDL i s'enllaça amb l'aplicació. Els llenguatges de programació orientats a objectes no necessiten *stubs*.

Vegeu l'IDL en el subapartat 3.2.5 d'aquest mòdul didàctic.

- **ORB al client**: com hem vist, l'ORB aïlla l'objecte client de l'objecte servidor i actua com a intermediari per a les requestes, de manera que no cal que el client i el servidor tinguin l'un informació sobre l'altre. L'ORB al client s'enllaça amb l'aplicació i verifica els arguments de la requesta i els compara amb els paràmetres de la interfície, i tramet la requesta al servidor. Dins una arquitectura CORBA hi pot haver més d'un ORB al mateix temps, amb estils d'invocació diferents i referències als objectes amb una estructura diferent, també; llavors, cada ORB ha de saber trobar la seva implementació d'un objecte demanat per un client, sense que aquest li hagi d'indicar.

- **Dipòsit d'interfícies**: conté les interfícies, i també les constants i definicions de tipus que es fan servir.

- **ORB al servidor**: rep les peticions d'execució de mòduls, dóna format als arguments i invoca l'execució del mètode a partir de l'esquelet. S'enllaça amb l'aplicació del servidor.

- ***Portable Object Adapter*** **(POA)**: els adaptadors d'objectes duen a terme tasques generals relatives a la implementació dels objectes, com activar els objectes i les seves implementacions a partir de les referències a aquests, desactivar-los, registrar les implementacions disponibles als servidors diferents, generar els identificadors dels objectes (si no se n'encarrega l'aplicació) i interpretar les referències als objectes mitjançant aquests identificadors; a tot això s'apliquen diverses opcions, que són les anomenades *polítiques de l'adaptador*. Els adaptadors d'objectes s'enllacen amb l'aplicació. En un moment determinat hi pot haver diversos POA formant una jerarquia per a un ORB, cadascun dels quals gestiona un grup d'objectes; d'aquests, el POA arrel

POA és la sigla de *Portable Object Adapter*.

Adaptador d'objecte en CORBA

Les primeres versions de CORBA especificaven un altre adaptador d'objectes en comptes del POA, el BOA (*Basic Object Adapter*).

s'engega automàticament i els altres s'engeguen segons la jerarquia per mitjà de l'activador de l'adaptador.

- **Aplicacions al servidor**: inclouen una implementació o més dels objectes i dels seus mètodes, i òbviament també codi per a la inicialització i finalització de la mateixa aplicació.

- **Mètodes**: cada mètode és el codi, contingut en la implementació d'un objecte, que implementa una determinada operació d'una interfície. El conjunt de les implementacions de les operacions d'una interfície –i, per tant, el conjunt dels mètodes corresponents– s'anomena **servent***. Per a una mateixa interfície hi pot haver diversos servents i, per tant, una operació d'una interfície pot ser implementada per diversos mètodes. La implementació dels objectes ha de ser independent de l'ORB.

*** En anglès, *servant*.**

- **Esquelets**: són correspondències, dependents del llenguatge de programació i també de l'ORB, entre les definicions d'operacions d'IDL i els mètodes corresponents i les implementacions que els contenen; inclouen la programació necessària per a llançar els mètodes imprescindible per a una requesta. Es generen en forma font a partir de la definició de la interfície respectiva, es compilen i s'enllacen amb l'aplicació del servidor. Els fan servir els adaptadors d'objectes. A més dels esquelets que estableixen relacions permanents entre operacions i mètodes, o esquelets específics del tipus, també hi pot haver esquelets dinàmics, que accedeixen en temps d'execució al codi de l'operació i als seus paràmetres.

- **Serveis d'objectes i serveis comuns**: es descriuen a part.

Vegeu els serveis d'objectes en el subapartat 3.2.9 i els serveis comuns en el subapartat 3.2.10 d'aquest mòdul didàctic.

- **Dipòsit d'implementacions**: conté informació que permet a l'ORB de localitzar i activar les implementacions d'un objecte quan s'hi fa una referència, de la mateixa manera que troba les interfícies demanades dins el dipòsit d'interfícies.

3.2.4. El processament de les requestes: visió resumida

El processament de les requestes té les fases següents:

1) El client invoca la requesta.

2) Quan arriba a l'ORB, la requesta inclou una referència a l'objecte destinatari d'aquesta. L'ORB, fent servir el dipòsit d'implementacions, localitza el procés del servidor o l'engega si cal.

3) L'ORB localitza el POA corresponent a l'objecte destinatari de la requesta, o bé en demana la creació a l'activador de l'adaptador; si no ho aconsegueix, el client rebrà l'excepció *object_not_exist*.

4) El POA localitza o activa el servent de l'objecte de maneres diferents segons les polítiques vigents. Un cop el servent és actiu, el POA selecciona dins aquest el mètode corresponent a l'operació invocada dins la requesta.

5) Es localitza l'esquelet.

6) Es fa la gestió de les respostes i les excepcions.

3.2.5. L'IDL

Les interfícies utilitzades dins l'entorn de CORBA es defineixen mitjançant la ***Interface Definition Language*** (IDL).

Altres llenguatges IDL

Dins mateix de la tecnologia de sistemes distribuïts, hi ha altres llenguatges que també s'anomenen IDL.

L'IDL no és un llenguatge de programació, i no està pensat perquè es pugui generar codi objecte o executable a partir de si mateix; però les implementacions de CORBA acostumen a poder generar codi font –principalment en C++, però també en llenguatges com COBOL, Smalltalk i Java– a partir d'aquest. La compilació d'una interfície definida amb IDL genera tres sortides:

- un ***stub***;
- un **esquelet**;
- un **fitxer de capçalera**, que conté definicions de tipus de dades com estructures i constants, i s'inclou dins les aplicacions dels clients i dels servidors.

A més, la interfície compilada queda dipositada en l'anomenat **dipòsit d'interfícies***.

*** En anglès, *interface repository*.**

La gramàtica de l'IDL és un subconjunt de la norma que s'ha proposat sobre el C++ ANSI, ampliat per a suportar el mecanisme d'invocació de requestes. L'IDL és un llenguatge declaratiu i conté la sintaxi del C++ per a la declaració de constants, tipus i operacions (si bé en alguns aspectes la sintaxi de l'IDL és més restrictiva).

Les especificacions d'IDL

Els tipus de valors...

... són tipus amb possibilitat d'herència, els valors dels quals no tenen identitat.

Una especificació d'IDL és un fitxer font, que pot contenir definicions de mòduls, interfícies, tipus de valors, constants i excepcions.

Els **mòduls** agrupen definicions d'interfícies i serveixen principalment per a definir àmbits de validesa dels noms dins una especificació; en general, les referències a noms d'elements definits dins un mòdul han de portar com a prefix el nom del mòdul. Hi ha un mòdul especial, CORBA, que reuneix tots aquells noms que formen part de les especificacions de CORBA.

3.2.6. Connexió d'una aplicació a l'entorn CORBA

Perquè una aplicació pugui entrar en l'entorn de CORBA cal fer abans el següent:

- Inicialitzar-la dins l'entorn d'un ORB o més –primer– i després potser també dins l'entorn de l'adaptador d'objectes respectiu.

- Obtenir referències al pseudoobjecte de l'ORB i eventualment també a altres objectes com el POA arrel, el POACurrent, el dipòsit d'interfícies i alguns serveis d'objectes, per a poder-ne demanar operacions posteriorment.

3.2.7. Les invocacions dinàmiques

Les invocacions dinàmiques són invocacions de requestes en què la requesta es crea en temps d'execució, a diferència de les invocacions estàtiques en què la requesta queda determinada en temps de compilació. En la invocació dinàmica de requestes es poden fer servir dos modes de comunicació: síncron i síncron diferit.

Una invocació dinàmica gestionada per CORBA té típicament els passos següents:

1) Obtenció de la informació sobre la interfície de l'objecte.
2) Creació de l'estructura de dades que es passarà a l'objecte.
3) Creació d'una requesta per a l'objecte.
4) Invocació de la requesta.

3.2.8. La interfície d'esquelets dinàmics (DSI)

Quan es fan servir esquelets dinàmics, a diferència dels específics d'un tipus, el nom de l'operació i fins i tot el tipus de l'objecte no es coneixen fins a l'execució.

En certa manera, és un cas simètric, de la banda del servidor, del que és la invocació dinàmica de la banda del client: de la mateixa manera que un servidor no sap si el client ha emprat invocació estàtica o dinàmica, un client no sap si el servidor fa servir un esquelet específic del tipus o bé dinàmic.

Juntament amb la invocació dinàmica, els **esquelets dinàmics** són una manera d'aconseguir ponts genèrics entre ORB.

Altres utilitats dels esquelets dinàmics són les eines interactives de desenvolupament del programari basades en intèrprets, les eines de prova i els monitors que s'interposen dinàmicament entre els objectes, i els llenguatges de tipificació dinàmica, com LISP.

Per a totes les requestes a un mateix objecte es fa servir la mateixa rutina, anomenada *rutina d'implementació dinàmica* (DIR); per a un mateix llenguatge de programació totes les crides a la DIR tenen els mateixos arguments.

DIR és la sigla de l'expressió anglesa corresponent a *rutina d'implementació dinàmica*.

3.2.9. Els serveis d'objectes

Característiques comunes

Tot i no formar part de l'entorn CORBA en sentit estricte, la manera com es fan servir els objectes en les especificacions dels serveis d'objectes respecta molts dels seus principis:

- Es fan servir interfícies per a tipificar els objectes.
- Hi ha una separació clara entre la interfície i la implementació; els clients veuen només les interfícies, no pas les seves implementacions.
- S'utilitza l'herència –fins i tot múltiple– entre interfícies per a estendre i especialitzar la funcionalitat, i permetre'n l'evolució.

A més, el disseny dels serveis d'objectes respecta, en línies generals, aquests principis:

- Són serveis genèrics, en el sentit que són independents del tipus del client i també –en general– del tipus de les dades passades en les requestes.
- Generalment, estan implementats en forma d'objectes de CORBA que poden ser activats localment o remotament.
- Sovint les interfícies permeten implementacions amb qualitats de servei diferents.
- Molts serveis tenen interfícies diferents per a tipus de clients diferents.

- Els serveis fan servir sovint interfícies *callback*, és a dir, fan requestes al client mitjançant interfícies que aquest ha de suportar, així aprofita també els avantatges de les interfícies en les crides al client.

- Les excepcions serveixen només per a comunicar situacions anòmales; els codis de retorn normals es passen al client per mitjà de paràmetres de sortida.

- Generalment, hi ha operacions diferents en comptes de modalitats diverses d'una operació diferenciades per un paràmetre d'indicadors*, per exemple.

*** En anglès, *flags*.**

El servei de noms

El servei de noms gestiona estructures de noms d'objectes que serveixen per a localitzar-los des d'altres objectes.

Un objecte pot tenir noms diversos. Una associació entre un objecte i el seu nom és un *name binding*. Un **context de noms*** és un espai de noms dins el qual els noms dels objectes no es poden repetir.

*** En anglès, *naming context*.**

Un context de noms, que és un objecte, té nom dins un altre context de noms, i així es poden constituir **jerarquies** –eventualment distribuïdes– de noms que donen lloc a noms compostos, constituïts per una seqüència de noms*, en què tots, excepte l'últim (el nom simple), són noms de contextos de noms. Un component consta d'un atribut identificador i un atribut de classe**, que fa una tipificació dels noms; aquesta tipificació no té valor sintàctic, per tal de permetre noms lligats a llenguatges de programació o idiomes concrets.

*** Els components del nom compost.**
**** En anglès, *kind attribute*.**

El servei d'esdeveniments

Pel que fa a CORBA, un **esdeveniment** és un fet que té relació amb un objecte i té interès per a altres objectes, als quals es fa avinent mitjançant un missatge anomenat *notificació*.

La notificació no la fa l'objecte –que no té per què saber que hi ha objectes interessats a conèixer que s'ha produït l'esdeveniment– sinó el servei d'esdeveniments.

Aquest servei distingeix dos tipus d'objectes: **subministradors***, que generen esdeveniments, i **consumidors**, que els processen. A més, hi pot haver un tercer tipus d'objectes, els **canals d'esdeveniments**, que són consumidors i subministradors alhora i permeten que diversos subministradors es comuniquin amb diversos consumidors sense que es coneguin entre si.

*** En anglès, *suppliers*.**

Hi ha els **esdeveniments amb tipus***, que fan possible que les aplicacions descriguin el contingut dels esdeveniments per mitjà de l'IDL, amb paràmetres només d'entrada i sense retorn. En aquest cas, hi pot haver dos canals d'esdeveniments en sèrie, un dels quals pot filtrar els esdeveniments per a l'altre basant-se en el tipus.

*** En anglès, *typed events*.**

Per a la **notificació** hi ha dos mecanismes:

1) ***Push***, en què el subministrador pren la iniciativa de transmetre informació sobre l'esdeveniment als consumidors, ja sigui directament o mitjançant un canal d'esdeveniments; el consumidor es pot "subscriure" als esdeveniments d'un tipus determinat i també pot decidir deixar de rebre esdeveniments.

2) ***Pull***, en què és el consumidor qui demana la informació sobre l'esdeveniment al subministrador, ja sigui directament o mitjançant un canal d'esdeveniments; el consumidor pot preguntar periòdicament pels esdeveniments, mentre que el subministrador pot registrar l'identificador del consumidor i oferir els seus serveis, i també pot deixar d'acceptar requestes sobre esdeveniments.

Quan la notificació té lloc mitjançant un canal d'esdeveniments, també pot ser mixta: *push* entre consumidor i canal i *pull* entre canal i subministrador o a l'inrevés.

Els consumidors i subministradors s'han d'haver posat d'acord quant a la semàntica dels esdeveniments, però el canal d'esdeveniments no la coneix.

El servei de relacions

El servei de relacions permet de crear relacions, permanents i temporals, entre dos objectes o més sense modificar-los i sense que aquests ho sàpiguen. El concepte de *relació* en aquest servei és semblant al d'*associació* en UML.

Hi ha tres nivells de servei:

1) El nivell de base defineix relacions i papers.

2) El nivell de grafs considera grafs d'objectes relacionats (això és, connectats per relacions); s'hi defineixen objectes nodes i objectes de recorregut d'un graf*, que permeten de recórrer un graf sense activar-ne els objectes.

*** En anglès, *traversal objects*.**

3) El tercer nivell considera dos tipus específics de relacions: per contingut i per referència.

El servei de cicle de vida

El servei del cicle de vida proporciona operacions per a crear, copiar, moure i esborrar objectes de manera local o remota; a més, suporta associacions entre grups d'objectes (ja siguin per contingut o per referència) i condicions d'integritat referencial entre objectes.

Per a crear un objecte cal que el client trobi un **objecte factoria** –és a dir, un objecte que sap com es crea un objecte de la classe corresponent–, que emeti una requesta de creació i que obtingui l'identificador de l'objecte creat. L'objecte factoria ha d'assignar els recursos necessaris, obtenir l'identificador de l'objecte i registrar el nou objecte mitjançant l'adaptador d'objectes i el dipòsit d'implementacions. També es pot crear un objecte copiant-lo.

El servei de propietats

El servei de propietats serveix per a definir atributs dels objectes dinàmicament, en contrast amb les interfícies d'IDL que ho fan estàticament.

Els atributs dinàmics, o propietats, tenen un nom, un tipus i un valor que es pot llegir i modificar, i també un mode de propietat, que pot prendre aquests valors:

- *normal*, és a dir, sense restriccions;
- *read-only*, que només deixa llegir i esborrar;
- *fixed-normal*, que deixa modificar però no esborrar;
- *fixed-readonly*, que només deixa llegir.

El servei de temps

El servei de temps serveix per a obtenir l'hora, confirmar l'ordre en què s'han produït esdeveniments diferents, generar esdeveniments relatius al temps i calcular el temps transcorregut entre dos esdeveniments.

La representació de temps UTC

El servei de temps fa servir la representació del temps anomenada *Universal Time Coordinated* (UTC) definida dins l'especificació *X/Open DCE Time Service*, en què la unitat són cent nanosegons i la base són les zero hores (hora de Greenwich) del 15.10.82.

El servei de temps integra dos serveis:

- El servei bàsic de l'hora*: comprèn operacions per a obtenir i manipular l'hora.

* En anglès, *Basic Time Service*.

- El servei d'esdeveniments de temporització*: proporciona operacions per a implementar gestors d'esdeveniments disparats per temps i gestionar els esdeveniments que generen.

* En anglès, *Timer Event Service*.

El servei d'externalització

Externalitzar un objecte vol dir convertir-lo en un objecte *stream* i **internalitzar-lo** vol dir fer el contrari; el servei d'externalització serveix per a totes dues coses.

Un objecte *stream* és una àrea de dades amb un cursor, que està en memòria o en disc o s'envia a la xarxa.

L'externalització serveix per a facilitar l'exportació/importació d'un objecte d'un procés, ordinador o ORB a un altre (o al mateix procés), en el qual es farà una internalització per a crear un objecte nou.

El servei d'objectes persistents

La funció del servei d'objectes persistents és desar l'estat d'un objecte en memòria permanent i recuperar-lo. Ho fa mitjançant una interfície que no té en compte el mecanisme d'emmagatzematge*.

*** Per exemple, un SGBD orientat a objectes, un de relacional o un fitxer.**

El client pot estar més o menys involucrat en el mecanisme d'emmagatzematge, o bé gens; en el segon cas, quan accedeix a un objecte, el fet que ja estigui en memòria o que hagi de ser recuperat de l'emmagatzematge permanent li és transparent.

El servei de control de la concurrència

La finalitat del servei del control de la concurrència és arbitrar els accessos concurrents a un objecte amb vista a garantir-ne la integritat. Proporciona interfícies perquè els clients puguin reservar i alliberar recursos per a coordinar-se en llur ús compartit.

Aquest servei pot funcionar en dos modes:

- **Transaccional**, és a dir, en representació d'una transacció; llavors, el servei de transaccions s'encarrega de l'alliberament dels recursos reservats quan la transacció acaba, ja sigui normalment o anormalment.

- **No transaccional**, això és, en representació del fil* que s'executa en cada moment, que no té per què ser una transacció; aleshores, és el mateix servei de control de la concurrència el que s'encarrega de fer els alliberaments quan cal.

*** En anglès, *thread*.**

Està dissenyat per a funcionar en combinació amb el servei de transaccions d'objectes i pot suportar transaccions encaixades.

No es defineix què és un recurs; és responsabilitat dels clients d'aquest servei tant definir els recursos com identificar els usos potencialment conflictius d'aquests; en el cas més típic, els recursos són objectes.

És possible que hi hagi recursos de diversos nivells o granularitats, és a dir, recursos que en contenen d'altres, però el servei no veu aquestes jerarquies. Es pot optar entre definir pocs recursos d'alt nivell o molts de baix nivell; en el primer cas cal fer menys reserves, però els conflictes seran més freqüents*.

*** És com reservar fitxers sencers en comptes d'enregistraments individuals.**

Tampoc es defineix què és cada transacció; això ho fa el servei de transaccions. Es considera que en els clients transaccionals cada transacció té un sol fil i que una transacció no s'executa per a més d'un fil alhora.

Un sol fil per transacció...

… no impedeix el paral·lelisme, ja que hi ha suport per a transaccions encaixades (vegeu el servei de transaccions, més endavant dins aquest mateix subapartat).

Una **reserva** (***lock***) és la capacitat d'un client concret d'accedir a un recurs determinat d'una certa manera; una reserva correspon, doncs, a un client i un recurs; un client ha d'aconseguir una reserva damunt un recurs abans de poder-hi accedir; com que hi ha reserves de diversos modes, el client ha de demanar la reserva d'un mode que li permeti de fer les activitats previstes damunt el recurs.

L'arbitratge esmentat en la definició del servei consisteix a evitar que diversos clients tinguin alhora reserves del mateix recurs si les activitats d'aquests clients podrien entrar en conflicte. El servei concedirà una reserva a un client només si cap altre client no té una reserva d'un mode incompatible amb el de la reserva que es demana ara.

Pel que fa a l'alliberament de recursos reservats*, si es demanen reserves dins una transacció, el servei de transaccions els allibera quan acaba la transacció; si es demanen reserves fora de transaccions, el client ha d'alliberar explícitament els recursos reservats. Sovint, les reserves es mantenen fins al final de la transacció, però en el cas que una transacció no modifiqui el recurs el pot alliberar quan ja sap que no el necessitarà més.

*** En anglès, *unlocking*.**

Locksets

Cada *lockset* és un conjunt de reserves sobre un mateix recurs; els clients han d'assignar un *lockset* a cada recurs.

Hi ha uns coordinadors de reserves*, cadascun dels quals pot gestionar *locksets* sobre recursos d'un mateix tipus que hagin de ser alliberats quan s'acaba una

*** En anglès, *lock coordinators*.**

mateixa transacció. La creació de *locksets* i l'alliberament de recursos quan una transacció confirma o avorta és responsabilitat del client.

El servei de transaccions

Les transaccions són unitats de procés que o bé arriben a acabar normalment, o bé, si avorten, les actualitzacions a bases de dades que haguessin fet s'anul·len i és com si ni haguessin començat.

El servei de transaccions té les funcions següents:

- Controlar l'abast i durada de les transaccions.
- Permetre que participin diversos objectes en una sola transacció.
- Permetre que aquests objectes associïn llurs canvis d'estat a una transacció.
- Coordinar l'acabament de les transaccions.

El servei de transaccions suporta dos models de transacció:

- **Transaccions encaixades** de qualsevol nombre de nivells. Una subtransacció pot confirmar o anul·lar (a petició de qualsevol dels objectes que hi participen) sense que necessàriament ho facin les subtransaccions que la contenen, però la seva confirmació no serà definitiva mentre la transacció no confirmi. Una transacció només pot confirmar quan ho hagin fet totes les seves subtransaccions i, quan la transacció és anul·lada, també ho són totes les seves subtransaccions.

- **Transaccions planes** segons el model DTP de l'X/Open, que no poden tenir subtransaccions.

El servei de seguretat

El servei de seguretat té aquests aspectes:

- Confidencialitat: que només tinguin accés a la informació els usuaris autoritzats.

- Integritat: que la informació només pugui ser modificada pels usuaris autoritzats i de la manera autoritzada.

- Responsabilitat*: que els usuaris siguin responsables de les seves accions en relació amb la seguretat.

- Disponibilitat: que l'accés al sistema no es pugui negar injustificadament als usuaris.

*** En anglès, *accountability*.**

Les funcions d'aquest servei són les següents:

- Identificació: demanar al principal (el principal és una persona o objecte titular d'una autorització d'accés) que digui qui és.
- Autenticació: comprovar que el principal és qui diu que és.
- Autorització dels principals.
- Control d'accés: determinar si s'ha de permetre l'accés d'un principal (prèviament identificat i autenticat) a un objecte, a partir dels atributs de l'un i l'altre.
- Auditoria de seguretat: determinar els humans responsables principals de les accions en matèria de seguretat, i identificar-los fins i tot a través d'una cadena de requestes.
- Seguretat de la comunicació entre objectes, que requereix l'autenticació mútua de client i servidor.
- No-repudiació: subministrar proves irrefutables de l'origen de les dades i de llur rebuda pel destinatari.
- Gestió de les polítiques de seguretat.

El servei té aquests nivells de funcionalitat:

1) El nivell 1, que proporciona serveis de seguretat a aplicacions que no en tenen coneixement i a aplicacions que tenen unes necessitats reduïdes de control i auditoria de la seguretat; comprèn la seguretat de les invocacions, la protecció dels missatges, algunes possibilitats de delegació, control i auditoria.

2) El nivell 2, que permet les aplicacions de controlar la seguretat en les invocacions d'objectes i suporta una gestió de seguretat portable.

3) La funcionalitat opcional, de la qual les especificacions actuals només inclouen la no-repudiació.

El servei de col·leccions

Les **col·leccions** són tipus diferents d'agrupacions d'objectes (elements).

Típicament, els elements d'una col·lecció o bé són del mateix tipus o bé tenen una mateixa interfície. Els diferents tipus de col·leccions difereixen quant a l'existència d'una ordenació dels objectes, l'existència d'accés als objectes per

clau, l'existència d'un criteri d'igualtat dels objectes i si hi pot haver o no diversos objectes amb el mateix valor de la clau; diferents combinacions d'aquestes restriccions donen lloc a tipus diferents de col·leccions; a cada tipus de col·lecció li correspon una interfície, i totes aquestes constitueixen una jerarquia de derivació, dins la qual tots els nivells llevat del més baix són d'interfícies abstractes.

Per a recórrer els objectes de les col·leccions hi ha uns cursors anomenats *iteradors*, dels quals n'hi ha un tipus diferent per a cada interfície abstracta de col·leccions.

El servei de consultes

Les **consultes*** no estan limitades a accessos només de lectura, com podria semblar, sinó que són instruccions declaratives amb predicats, que poden comprendre valors d'atributs i invocacions d'operacions i altres serveis d'objectes.

*** En anglès, *queries*.**

El resultat d'una consulta pot ser o bé una col·lecció obtinguda seleccionant aquells objectes d'una col·lecció font (que també podria ser el resultat d'una consulta anterior) que compleixen un predicat donat o bé pot ser generat per un **avaluador de consultes** a partir d'un predicat que s'avaluaria damunt una col·lecció virtual.

El servei de consultes pot coordinar diversos avaluadors encaixats i federats.

Els objectes poden participar en el servei de dues maneres:

1) A títol individual; aleshores l'avaluador de consultes s'encarrega d'avaluar el predicat de la consulta i de dur a terme totes les operacions de la consulta per mitjà d'operacions definides en la interfície corresponent als objectes. Aquest és el mecanisme més general però també el menys optimitzat. És en aquest cas que els objectes afectats constituirien una col·lecció virtual de les que hem esmentat.

2) Com a elements d'una col·lecció, la qual suporta una interfície per a consultes; l'avaluador li passa el predicat i aquesta l'avalua, fa les operacions damunt els objectes individuals, en combina els resultats i tramet el resultat a l'objecte que l'havia invocat. Aquest mecanisme permet que els avaluadors* apliquin els seus instruments d'optimització.

*** Per exemple, l'SGBD.**

L'especificació del servei de consultes no defineix mecanismes d'avaluació, indexació ni optimització, els quals es deixen a criteri de l'implementador; però sí que preveu llenguatges de consulta concrets, que són els següents:

- SQL-92 Query i els seus successors.
- OQL-93 i OQL-93 Basic, de l'ODMG, i els seus successors.

Algunes característiques destacades del servei de consultes són les següents:

- Proporciona operacions per a seleccionar, inserir, actualitzar i esborrar elements dins de col·leccions.
- Els elements afectats poden ser objectes persistents o transitoris, locals o remots.
- En els predicats es poden fer servir atributs, herència i navegació mitjançant relacions, tot per mitjà de les interfícies dels objectes elements.

El servei de llicències

En el món dels objectes distribuïts hi pot haver objectes que siguin de pagament; llavors convé que es pugui mesurar l'ús dels objectes amb vista a una facturació posterior. El servei de llicències dóna suport a aquesta funció.

Es consideren els tipus de llicències següents: amb període de gràcia, amb llistes d'usuaris inclosos o exclosos, amb llicències reservades sempre disponibles i amb llicències multiús. Des d'un altre punt de vista, llicències per a una màquina*, llicències a nivell d'instal·lació i llicències flotants per a un nombre màxim d'usuaris concurrents.

*** En anglès, *nodelocked*.**

El servei de llicencies ha de complir unes característiques necessàries per a optimitzar-ne el rendiment:

- Possibilitat de creixement.
- Mesures per a evitar que quedin llicències assignades indefinidament a una aplicació.
- Garantia que només es facin servir les llicències comprades.
- Prevenció de clients i servidors impostors.

Un **productor*** és una empresa o persona que té la propietat intel·lectual, l'ús de la qual es vol controlar.

*** En anglès, *producer*.**

Un **client del productor*** és qualsevol objecte del qual s'ha de controlar l'ús que fa d'una llicència.

*** En anglès, *producer client*.**

Una **política del productor*** és un conjunt de dades que descriu els termes i condicions detallats que regeixen el control de l'ús d'una llicència.

*** En anglès, *producer policy*.**

Un **document de llicència** proporciona un mitjà per a especificar les limitacions a l'ús de la llicència: nombre d'exemplars de la propietat intel·lectual, limitacions temporals, etc.

El servei de l'intermediari d'objectes (*Object Trader*)

Un objecte que presta uns serveis determinats (**exportador**), demana de ser registrat* i, per això, per a cada servei en comunica l'**oferta de servei**, que conté el nom del tipus de servei, una referència a la interfície que proporciona el servei i els valors de les propietats del servei.

*** Llavors es diu que l'objecte exporta aquests serveis.**

La informació sobre cada tipus de servei amb què tracta un objecte intermediari determinat comprèn un tipus d'interfície i, opcionalment, un tipus de propietat o més. Un tipus de servei pot ser subtipus d'un altre.

Es retorna un identificador de l'oferta a l'exportador, que li permet de modificar-la o retirar-la posteriorment.

Els clients (**importadors**) poden obtenir una llista de serveis disponibles (importar-los), en general, o bé per tipus com a les pàgines grogues. Per això han d'indicar el tipus de servei desitjat i una restricció especificada en el llenguatge de restriccions*, els elements principals del qual són: tipus de valors de propietats, operadors i literals.

*** En anglès, *standard constraint language*.**

L'intermediari cerca el servei que s'ajusta més bé a allò que demana el client, però és aquest qui interactua directament amb el proveïdor que tria. Aquest fa servir el tipus de servei, la restricció especificada i les preferències que han de servir per a establir un ordre de presentació a l'importador de les ofertes seleccionades.

Unes **polítiques** permeten d'identificar el conjunt d'ofertes de serveis dins el qual ha d'efectuar la cerca. Les polítiques tenen un nom i un valor.

Es poden crear federacions d'objectes intermediaris i, per tant, dels dominis de tipus de servei o particions, dins l'àmbit de les quals es propaguen les consultes entre intermediaris.

3.2.10. Els serveis comuns (*Common Facilities*)

Els serveis comuns són serveis que poden ser compartits per les aplicacions i que són d'una naturalesa menys bàsica que els serveis d'objectes. En la seva implementació intervenen estructures d'interfícies.

Els serveis comuns es divideixen en els següents:

1) **Serveis horitzontals**, que són utilitzables a la majoria d'aplicacions.

Els serveis comuns horitzontals són els següents: la interfície amb l'usuari, gestió de la informació, gestió del sistema i gestió de tasques.

Terminologia en anglès

1) Gestió de la informació en anglès és *Information Management.*

2) Gestió del sistema en anglès és *System Management.*

3) Gestió de tasques en anglès és *Task Management.*

2) Serveis verticals, que són específics d'un domini o sector empresarial.

Els serveis verticals que s'han especificat fins ara a l'àmbit de CORBA són els següents: medicina, telecomunicacions, finances i fabricació.

Només veurem els serveis horitzontals.

La interfície amb l'usuari

La interfície amb l'usuari fa servir una tecnologia de documents compostos i serveis d'edició semblants als d'OLE. L'objectiu és subdividir la finestra de manera que la puguin compartir diversos objectes; per a això cal que els objectes segueixin protocols determinats. Es fan servir *scripts*, cosa que permet que els components del document es combinin dinàmicament, és a dir, en temps d'execució i no pas de compilació. Comprèn els serveis següents:

- Gestió del *rendering* (presentació general dels objectes).
- Presentació de documents compostos.
- Suport de l'usuari (informació d'ajuda i verificació de text).
- Gestió de l'escriptori*.
- Creació interactiva de *scripts*.

*** En anglès, *desktop*.**

La gestió de la informació

La gestio de la informació suporta l'emmagatzematge de documents compostos i l'intercanvi de dades de manera semblant a OLE. Comprèn els serveis següents: modelització de la informació, amb un model semblant al de l'IDL, emmagatzematge i recuperació de la informació, intercanvi d'informació (dades i operacions entre dispositius), de documents compostos i de dades, codificació i representació de les dades i operacions de temps (comparacions entre hores i intervals).

Vegeu el model IDL en el subapartat 3.2.5 d'aquest mòdul didàctic.

La gestió del sistema

La gestio del sistema proporciona interfícies per a les funcions de gestió del sistema: control de monitoratge, gestió de la seguretat, configuració i altres que puguin fer falta per a afegir usuaris, donar autoritzacions, instal·lar programari, etc.

La gestió del sistema considera els tipus d'usuaris següents: el gestor del sistema, els desenvolupadors d'aplicacions de gestió, els proveïdors de serveis del sistema i els planificadors dels recursos del sistema.

Els recursos del sistema...

... poden ser físics, com impressores i encaminadors (*routers*), o lògics, com usuaris, grups i aplicacions.

La gestió de tasques

La gestió de tasques ofereix una infraestructura per a les aplicacions que donen suport a les tasques de l'usuari, mitjançant un servei de missatges d'alt nivell

entre tasques. Comprèn els serveis següents: fluxos de treball*, agents, fixos i mòbils, gestió de regles i automatització.

* En anglès, *workflows*.

3.2.11. Interoperabilitat entre ORB

La interoperabilitat entre ORB pretén de suportar la distribució i la intercomunicació d'objectes entre còpies i implementacions d'ORB diferents que compleixin les especificacions de CORBA.

Es pot dir que així com un ORB fa possible que els objectes enviïn i rebin requestes i respostes mitjançant aquests de manera transparent, la interoperabilitat entre ORB estén aquesta transparència al cas en què els objectes esmentats estan gestionats per ORB diferents.

La interoperabilitat està suportada pels elements següents: l'arquitectura de la interoperabilitat, el suport de ponts entre ORB i dos tipus de protocols interORB: els generals (GIOP) i els d'Internet (IIOP), a més de protocols específics per a determinats entorns, com el DCE (ESIOP).

L'arquitectura de la interoperabilitat

L'arquitectura de la interoperabilitat proporciona un marc conceptual per a identificar els elements de la interoperabilitat i definir els aspectes que han de complir. També estableix mecanismes i convencions per a aconseguir la interoperabilitat entre ORB produïts independentment; en particular, introdueix el concepte de *ponts interORB*.

En aquest context, un **domini** és un conjunt d'objectes (els membres del domini) que tenen una característica en comú.

Un objecte pot pertànyer a diversos dominis. Un domini és també un objecte. Hi ha dos tipus de dominis:

- **administratius**, com ara els dominis de noms o aquells que estan lligats a recursos;
- **tecnològics**, lligats a protocols, per exemple.

Entre dominis hi pot haver relacions de dues menes:

- **de conteniment**, quan l'àmbit d'un és subconjunt del de l'altre;
- **de federació**, quan els dominis s'ajunten per acord dels gestors respectius.

Els ponts entre dominis

L'interès del concepte de *domini* per a la interoperabilitat està en el fet que aquesta consisteix precisament a superar les fronteres dels dominis, cosa que s'aconsegueix mitjançant unes correspondències o ponts entre dominis que permetin de transformar les requestes expressades en termes d'un domini en requestes expressades en termes de l'altre. Com que el concepte *d'interoperabilitat* és simètric, cal que les transformacions que els ponts suporten siguin bidireccionals.

Ponts segurs entre dominis

No sempre és desitjable la transparència total, ja que de vegades s'estableixen dominis precisament per raons de seguretat o de gestió de recursos compartits o compartibles; en aquests casos, el pont ha d'incorporar els filtres escaients.

Els **ponts** permeten que els ORB col·laborin sense haver de tenir en compte els detalls de la implementació de l'altre; també hi pot haver ponts que permetin la interoperabilitat amb sistemes no basats en CORBA, com és el cas de COM de Microsoft. A més, els ponts poden servir per a determinades situacions transitòries*, generació automàtica d'implementacions per a un ORB a partir d'implementacions fetes per a un altre, etc.

*** Transitòriament es pot començar a fer servir un nou ORB que coexisteixi amb l'antic.**

Hi ha dues menes de ponts:

a) Immediats, en què els elements d'un domini es transformen directament en els de l'altre; és una opció adequada quan el canvi de domini és purament administratiu, és a dir, no hi ha canvi de tecnologia.

b) Intermediats; en aquests darrers, el que es construeix són "mitjos ponts" que fan transformacions entre els formats interns dels dos dominis i un tercer format; és una opció més flexible però menys eficient que l'anterior.

Pel que fa a llur relació amb els ORB, els ponts es poden implementar de dues maneres: internament a l'ORB (*in-line*), ja sigui com a serveis o com a codi d'*stubs* i esquelets; i com a capes per damunt d'aquest (*request-level*).

Les referències interoperables a objectes

Les referències interoperables a objectes presenten les premisses següents:

1) No cal que els clients sàpiguen si les referències dels objectes als quals adrecen peticions són locals o remotes, ni si són del mateix ORB o d'un altre.

2) No cal que un ORB pugui tractar les referències a objectes assignades per un altre ORB.

Per tant, en principi hi ha un domini de referències a objectes per cada ORB i cal preveure el pont corresponent. També cal preveure la conversió de codis de caràcters.

El protocol interORB general (GIOP)

El protocol interORB general serveix per a la comunicació directa entre ORB, i està pensat per a funcionar directament damunt de qualsevol protocol orientat a la connexió. Consta dels elements següents:

1) La *Common Data Representation* (CDR), que és una sintaxi estàndard que transforma tipus de dades d'IDL en una representació de baix nivell per a la transferència entre ORB o bé entre ponts interORB.

CDR és la sigla de *Common Data Representation.*

2) Un conjunt de formats de missatges que poden ser de client, de servidor o mixtos, per a facilitar les requestes, localitzar implementacions d'objectes (fins i tot quan poden migrar dinàmicament) i gestionar canals de comunicacions, a més de suportar tota la funcionalitat de CORBA entre ORB.

3) Un conjunt de requisits quant al transport: ha de ser orientat a la connexió, ha de garantir que els bytes arribin en l'ordre en què s'han enviat i sense repeticions, tampoc no hi ha d'haver límit en la llargada dels missatges, ni ha de caldre fragmentació o alineació.

El protocol interORB d'Internet (IIOP)

El protocol interORB d'Internet forma part de l'especificació del GIOP. Especifica com es poden bescanviar missatges de GIOP mitjançant el TCP/IP, i també es pot fer servir com a protocol entre ponts intermediats ("mitjos ponts"). Les definicions del GIOP són abstractes, i l'IIOP les converteix en concretes; així l'IIOP descriu l'ús de les connexions de TCP/IP i de quina manera s'identifiquen els objectes.

Els protocols interORB per a entorns específics (ESIOP)

CORBA preveu que hi pugui haver protocols per a la interoperació amb infraestructures concretes de xarxa o de computació distribuïda preexistents, per als quals poden suportar funcions específiques com la seguretat i administració; aquests protocols haurien de complir les regles generals per a la interoperabilitat. La versió 2.3 de CORBA inclou l'especificació d'un ESIOP per a DCE, DCE *Common Inter-ORB Protocol* (DCE-CIOP).

Ara convé que feu els exercicis d'autoavaluació del 5 al 14 d'aquest mòdul didàctic.

4. RMI

RMI (*Remote Method Invocation*) és l'eina de Java per al suport d'objectes distribuïts.

RMI és la sigla de *Remote Method Invocation*.

RMI té una funcionalitat molt reduïda en comparació a CORBA, però en canvi, té l'avantatge que les invocacions remotes tenen lloc sense sortir de l'entorn Java. Aquí no es veuran les sentències de Java que fan aquesta funció, sinó només els seus aspectes generals.

Les invocacions remotes són com les locals des del punt de vista formal. De la mateixa manera que en aquestes darreres, doncs, el valor d'un argument o el resultat pot ser un objecte de qualsevol classe, de manera que no solament es poden transmetre dades entre els objectes client i el servidor, sinó també objectes complets i implementacions d'objectes* que es poden executar un cop rebudes. Per a passar els objectes es fan servir els mecanismes estàndard de serialització de Java.

*** Per exemple, agents.**

RMI es pot combinar amb JNI per a accedir a servidors programats en altres llenguatges, i amb JDBC per a accedir a bases de dades relacionals. Es poden aplicar les funcions de seguretat i la comprovació de tipus de Java.

Vegeu la documentació sobre Java de l'assignatura *Programació orientada a l'objecte*.

4.1. Mecanisme d'una invocació remota

Cal definir el tipus de les referències per a cada servidor que s'ha d'exportar. Quan un client rep una referència a un servidor RMI, obté un *stub* que dóna format als arguments, fa servir la serialització i envia la invocació al servidor. A la banda del servidor, RMI rep la invocació i la connecta a un esquelet que restitueix el format als arguments i invoca la implementació del mètode al servidor; un cop executat el mètode, l'esquelet dóna format al resultat –sigui un valor o una excepció– i envia el valor al client o provoca l'excepció. Tant els *stubs* com els esquelets es generen a partir de la implementació del servidor.

5. Documents compostos distribuïts: DCOM

Tot i que s'acostuma a considerar que la manipulació de documents pertany més al domini de l'ofimàtica que de la informàtica professional, els documents compostos ens interessen per dues raons:

1) Perquè dins una interfície gràfica d'usuari poden figurar documents, considerats objectes.

2) Perquè els documents compostos poden resultar de la combinació de documents distribuïts, i la tecnologia de documents distribuïts és un cas particular de la tecnologia d'objectes distribuïts.

5.1. Concepte de document compost

Els documents són objectes que es presenten directament a l'usuari de manera visual i/o auditiva.

> Els **documents compostos** són aquells documents que resulten de l'agregació d'altres documents (que també poden ser compostos), que anomenarem **components**, dins un determinat document marc.

Els objectes compostos acostumen a ser generats en temps d'execució; d'altra banda, objectes components diferents poden ser gestionats i presentats per aplicacions diferents; per tant, la característica bàsica que ha de tenir una eina de gestió de documents compostos és un protocol que permeti la comunicació entre un conjunt prou ampli d'aplicacions que gestionin menes diferents de documents. La utilització de les aplicacions diferents en qüestió és molt més còmoda per a l'usuari si es fa mitjançant un document compost que no pas si es fa passant d'una aplicació a l'altra de la manera convencional: passant d'una finestra a una altra, obrint manualment cada fitxer, etc.

5.2. Aspectes de la gestió dels documents compostos

A continuació presentem aspectes diferents de la gestió dels documents compostos:

a) Presentació: el document marc es presenta dins una sola finestra; qualsevol document compost ha de tenir l'aparença d'un document únic. L'aplicació que gestiona un document compost gestiona els contenidors on s'ubiquen els

documents components, i en particular comunica a les aplicacions que els gestionen tots els esdeveniments que tenen a veure amb el contenidor respectiu*.

* Per exemple, si es redimensiona.

b) Emmagatzematge estructurat: un document simple es desa en un fitxer monolític, mentre que un document compost es desa en un fitxer que té parts que corresponen als diferents documents components; la part corresponent a un document pot contenir, efectivament, el document o bé tenir un punter cap al document, que llavors resideix en un fitxer independent.

c) *Scripts*: són programes que s'executen cada vegada que es produeix un determinat esdeveniment*, i la seva funció pot ser, per exemple, demanar una clau d'accés, donar accés a més o menys parts del document segons la clau d'accés, accedir a un gestor de dades (*data warehouse*), etc. Val a dir que els *scripts* s'han de poder preparar d'una manera que no requereixi programació convencional, ja que generalment els usuaris que creen els documents no són programadors.

* Per exemple, quan es llegeix o grava un document.

d) Transferència de dades uniforme: els documents compostos han de poder intercanviar dades –documents, principalment, que inclouen documents compostos sencers– amb aplicacions exteriors*.

* Per exemple *cut and paste, drag and drop*, enllaç mitjançant punters, etc.

5.3. OLE, COM i DCOM

OLE és una ampliació del portapapers i de DDE per a fer documents compostos orientats a objectes.

DDE

Dynamic Data Exchange (DDE) permet de vincular un element component a diversos documents compostos, sense estar contingut en cap d'aquests i de manera que es pot editar en principi tant directament com dins un document compost.

La inclusió de documents components es pot fer de dues maneres:

- amb ***embedding*** (inclusió), una còpia de l'objecte i el seu format de presentació s'emmagatzema dins el document i es transfereix amb aquest;

- amb ***linking*** (enllaç), el document que s'inclou dins el document compost continua dins la seva ubicació original en un document exterior o dins el mateix document compost, i l'objecte que el representa dins el document compost només conté el format i un punter a aquell fitxer.

Evolució de la tecnologia d'integració de documents

La sigla OLE ve d'*Object Linking and Embedding*. El 1990, Microsoft va introduir la tecnologia OLE 1 com a eina bàsica per a integrar documents creats per diferents aplicacions i dades multimèdia dins un document marc; funcionava sobre la base que quan una aplicació necessitava un document creat per una altra, aquesta darrera s'engegava dins una finestra a part.

OLE 2, aparegut el 1993, incorporava com a nucli una tecnologia d'encapsulació d'objectes anomenada COM (*Component Object Model*); així, OLE 2 ja no era només una eina per a gestionar documents compostos, sinó una arquitectura per a objectes força més general. El 1994 s'hi va afegir un concepte nou, l'OCX, que ve a ser un component genèric que s'inclou dins el document marc; però ja no hi va haver un OLE 3 –des d'aleshores les versions noves d'OLE simplement formen part de les versions noves de Windows.

Posteriorment, ha aparegut DCOM (*Distributed Component Object Model*); DCOM estén COM a entorns LAN, WAN i àdhuc Internet. COM i DCOM han passat de Microsoft al consorci ActiveX, i DCOM està disponible per a Windows NT 4.0 i Windows 2000, i per a Windows 95 i 98; a més, Software AG n'ha fet implementacions per a diverses plataformes Unix: Solaris, Linux i HP/UX. Nogensmenys, l'estratègia de Microsoft no és que Windows funcioni necessàriament amb OLE, sinó que la interfície d'usuari del Windows pugui estar suportada també per productes com Taligent i OpenStep.

5.3.1. Arquitectura

El ***Component Object Model*** (**COM**) ve a ser un ORB per a un únic ordinador; DCOM suporta la distribució.

COM és la sigla de *Component Object Model*.

Dins l'arquitectura podem trobar el grup d'elements següents: **objectes**, **classes**, **interfícies** i **servidors**.

1) S'anomenen **components** els objectes d'OLE, és a dir, els documents que es poden incorporar a un document complex. Un objecte de COM* no és un objecte clàssic: és només un grup de funcions (anomenades també *mètodes* o *funcions de membres*) relacionades, però que no té estat ni identitat, i un client no pot tornar a connectar amb el mateix objecte, sinó solament a un punter d'interfície de la mateixa classe.

*** També es coneix com *objecte d'OLE* o *objecte de Windows*.**

En aquests subapartats els termes *objecte*, *component* i *document* es faran servir indistintament.

Un document compost és el resultat de la interacció entre contenidors i servidors; es podria dir que els contenidors són llocs i els servidors són les coses que es posen en aquests llocs.

Atenció

Aquí, el significat del terme *servidor* no és el mateix que quan es parla de *client/servidor*; tanmateix els contenidors es poden considerar clients dels servidors.

2) Les **classes** que implementen una interfície o més tenen un identificador únic que s'anumena *CLSID*; un objecte és una implementació en temps d'execució d'una classe. Un component consta d'una classe que implementa una interfície o més i una factoria de classe que instancia un objecte de la classe.

3) OLE i COM especifiquen **interfícies** entre objectes components dins una aplicació o entre aplicacions i proporcionen una API per a la localització dinàmica d'interfícies i per a carregar-les i executar-les; un component pot tenir una interfície o més. Hi ha interfícies del mateix OLE i interfícies d'usuari*.

*** En anglès, *custom interfaces*.**

Per a definir aquestes interfícies Microsoft té dos llenguatges: l'*Interface Definition Language* (IDL) i l'*Object Description Language* (ODL), que el conté. El llenguatge ODL per a la definició d'interfícies és un llenguatge textual que es pot codificar manualment o bé se'l pot generar per mitjà del Visual C++ de Microsoft. Les interfícies de COM no generen codi en llenguatges de programació, com les de CORBA, sinó solament una API en binari per a accedir a les interfícies mitjançant punters* i es dipositen en una biblioteca de tipus (*type library*), que té un paper semblant al del dipòsit d'interfícies de CORBA.

IDL és la sigla d'*Interface Definition Language*.
ODL és la sigla d'*Object Description Language*.

*** Els clients accedeixen a tots els objectes per mitjà de punters a interfícies.**

Les interfícies tenen noms que han de començar per *I*, però en temps d'execució s'identifiquen per un identificador d'interfície (IID), que és generat per COM i no repetit, i que també identifica l'objecte. Totes les interfícies de COM deriven de la interfície *IUnknown*.

Les interfícies de l'usuari consten de mètodes amb llurs paràmetres, es creen en IDL i es compilen amb el compilador de MIDL, i d'aquesta compilació s'obtenen servidors intermediaris (*proxies*) del client, *stubs* del servidor i el codi que fa correspondre els paràmetres d'uns i altres, tot això en C.

Proxies* i *stubs

Els *proxies* (servidors intermediaris) i els *stubs* es fan servir quan el client i l'objecte cridat estan en processos diferents.

Hi ha tant la invocació estàtica d'interfícies com la dinàmica, que pot fer servir una llibreria de tipus, on hi ha descripcions d'objectes en ODL precompilades, amb les seves interfícies i paràmetres.

4) Un **servidor** és un fitxer .dll o .exe que conté una classe o més; quan un client demana un objecte d'un determinat CLSID, COM demana al servidor de crear un objecte d'aquella classe, i per a això el servidor ha de proporcionar la factoria de classe; quan es crea un objecte, el servidor en retorna l'IID de la interfície primària.

L'herència

No hi ha herència múltiple, però se supleix sobre la base que els objectes puguin suportar interfícies diverses. Això permet de crear objectes que presentin als clients els serveis dels objectes que contenen agrupats (és a dir, que els encapsulin), cosa que es pot aconseguir amb dos mètodes:

- **Conteniment i delegació**: l'objecte "contenidor" envia als objectes interns les invocacions de llurs mètodes que rep dels clients.

- **Agregació**: la interfície de cada objecte intern forma part de la de l'objecte "contenidor" i, per tant, els clients es poden adreçar directament als objectes interns.

Esdeveniments

En COM el suport dels esdeveniments es fa per mitjà dels objectes o fonts* connectables, que suporten interfícies de sortida –a més de les d'entrada; cada funció d'una interfície d'aquestes correspon a un tipus d'esdeveniment COM que proporciona interfícies, anomenades *embornals***, perquè altres objectes es puguin subscriure als esdeveniments que les fonts envien codificats en ODL; no hi ha res semblant en els canals d'esdeveniments de CORBA.

* En anglès, *sources*.
** En anglès, *sinks*.

5.3.2. Els documents compostos i els OCX

Un **document compost d'OLE** conté tant dades pròpies com objectes gestionats per altres aplicacions, i típicament li correspon un fitxer compost i fa d'intermediari entre dos tipus de components: els contenidors, que proporcionen llocs on posar coses, i els servidors, que són les coses que s'hi posen. Un tercer tipus de components d'OLE són els **OCX**, que són components amb un conjunt d'interfícies predefinides.

Components OCX

En anglès, *custom controls*. L'extensió dels seus fitxers és .ocx.

Vegeu el concepte de *moniker* al subapartat 5.3.4 d'aquest mòdul didàctic.

Un **contenidor** és una aplicació d'OLE que conté un document compost (o bé físicament o bé només un punter a aquest en forma de *moniker* del servidor o servidors de l'objecte). Per a cada servidor el contenidor crea un lloc*, on actuarà el servidor i presentarà el seu contingut.

*** En anglès, *site*.**

Classificació dels contenidors i servidors

Els contenidors poden ser **purs**, que només suporten *embedding*, i ***linked***, que només suporten *linking*.

La classificació dels servidors és la següent:

a) ***In-process***: s'implementen en forma de DLL dins el mateix espai d'adreces que el contenidor i només poden tractar objectes *embedded*; se subdivideixen en els següents:

- *InProcHandlers*, en què diversos servidors poden compartir una mateixa finestra i menús relatius a un document.
- *InProcServers*, DLL locals utilitzades pels servidors locals.
- *Controls*.

b) Locals, que s'implementen en forma de fitxers .exe separats, es comuniquen per *Lightweight* RPC amb els contenidors i s'executen dins la mateixa màquina i sistema operatiu que llurs clients. Una classificació dels servidors per funció és la següent:

- *full servers*, que suporten tant *embedding* com *linking* i són aplicacions autònomes com l'Excel i el Word;
- *mini servers*, que només suporten *embedding* i només poden funcionar dins l'aplicació del contenidor, com és el cas del Graph de Microsoft.

c) Remots, que es comuniquen per mitjà d'RPC ordinàries amb els contenidors.

Els OCX

Un OCX és una combinació d'un servidor *in-process* i un servidor d'automatització, i suporta *embedding*, edició *in-place*, automatització, notificacions i a més objectes connectables, llicències i edició de propietats.

Els contenidors per als OCX són contenidors per a objectes *embedded* i edició *in-place* que a més tenen unes propietats d'entorn* accessibles a l'OCX i unes propietats esteses que gestionen per a aquest (a part de les propietats que gestiona el mateix OCX, que són les propietats del control).

*** Propietats com el color del fons, la mida del tipus de lletra, etc.**

5.3.3. La transferència de dades

El que es transfereix –mitjançant un protocol de transferència d'OLE– són objectes de dades, el contingut dels quals pot estar representat en tres **formats diferents**:

1) estàndard, que comprèn els formats utilitzats en el portapapers de Windows;
2) privats, que són específics de les aplicacions;
3) d'OLE, que poden tenir dues estructures diferents:

a) FORMATETC, que conté una descripció del tipus de dades, el dispositiu d'emmagatzematge i un punter a les dades.

b) STGMEDIUM, que descriu el mitjà utilitzat per a transferir les dades (memòria, disc o objectes de documents compostos d'OLE), i un punter a la variable que es fa servir per a fer-ho.

Hi ha tres protocols per a la transferència de dades: retallar/copiar i enganxar per mitjà del portapapers, *drag and drop* i mitjançant enllaços.

Transferència de dades mitjançant enllaços

L'enllaç entre documents inclou un mecanisme d'actualització automàtica de totes les còpies del document per "publicació i subscripció": es crea un objecte amb una còpia de les dades, que es diposita dins un directori al qual tenen accés els subscriptors i conté també els enllaços a aquests. Dins de cada document subscriptor hi ha un objecte que conté la còpia (el qual és únic fins i tot quan la còpia es fa servir en més d'un lloc del document); quan es modifica el document s'envia una notificació a tots els subscriptors, que llavors en poden demanar una còpia actualitzada.

5.3.4. L'emmagatzematge estructurat i els *monikers*

OLE té l'**arquitectura estructurada d'emmagatzematge** que permet de desar en un sol fitxer compost* diversos documents independents entre si. Els serveis de persistència i emmagatzematge estructurat d'OLE fan servir tres tipus d'elements:

*** En anglès, *compound file*.**

1) Els **fitxers compostos** o **objectes d'emmagatzematge** constitueixen una jerarquia de fitxers implementada en forma d'elements de directori anomenats *emmagatzematges**, i d'una mena de fitxers anomenats *streams* que contenen una porció de dades més un punter per a accedir-hi. Entre els emmagatzematges n'hi ha un que és l'arrel del fitxer compost i té una funció per a associar l'objecte

*** En anglès, *storages*.**

d'emmagatzematge a un fitxer dels gestionats pel sistema operatiu. Un objecte d'emmagatzematge no conté dades de l'aplicació, però sí noms d'emmagatzematges i de *streams*.

2) Els **objectes persistents** són objectes que es poden llegir o gravar a si mateixos. Els objectes poden suportar interfícies diferents –una o més alhora– que els donen nivells diferents de persistència.

3) Els ***monikers*** són objectes que actuen com a àlies persistents d'altres, com ara fitxers, consultes a base de dades, paràgrafs de documents, etc., i hi ha diverses classes definides aplicables a diferents menes d'objectes: de fitxers, d'*items* (fragments de documents), *anti* (que anul·len un *moniker* anterior), de punters a memòria, i compostos (que inclouen altres *monikers*, fins i tot de compostos); a més, DCOM fa servir *monikers* d'URL, per als protocols http, https, ftp i gopher, que permeten d'accedir a objectes persistents dins Internet.

5.3.5. Els *scripts* i les llibreries de tipus

Els programes clients poden invocar mètodes que manipulen objectes amb *scripts*. Per a això hi ha tres classes de components:

1) Servidors d'automatització, que són els objectes als quals dóna accés una aplicació amb *scripts*. N'hi pot haver més d'un per aplicació. Tenen **mètodes**, que són funcions de membres, i **propietats**, cadascuna de les quals és un parell de funcions de membres, una que posa un valor i una altra que el llegeix.

2) Controladors d'automatització, que són les eines i programes dels clients que accedeixen al servidor d'automatització. El Visual Basic, a partir de la versió 3.0, és un controlador d'automatització.

3) Una **llibreria de tipus** descriu els mètodes d'entrada i de sortida i les propietats d'un servidor.

4) Un sol programa pot controlar servidors d'automatització de diverses aplicacions.

5.3.6. La interoperabilitat entre CORBA i COM/DCOM

La interoperabilitat entre CORBA i COM/DCOM pretén permetre l'accés d'un client CORBA a un servidor DCOM i a l'inrevés.

Les especificacions de CORBA descriuen les correspondències entre diferents elements de CORBA i de COM: tipus de dades, identificadors d'interfícies, excepcions, operacions, atributs i herència perquè es puguin automatitzar.

Ara feu els exercicis d'autoavaluació del quinze al vint d'aquest mòdul didàctic.

6. Desenvolupament de programari distribuït

6.1. L'anàlisi de requisits en el cas de programari distribuït

L'anàlisi de requisits en el cas de programari distrïbuit és independent del fet que el programari que se'n derivi sigui distribuït o no; la raó és que aquesta anàlisi s'ocupa del que ha de sortir a l'usuari per les pantalles i per les impressores, independentment de si el procés es fa tot localment o no.

6.2. La distribució dels objectes

En el cas de programari distribuït el disseny és igual que en el cas general, llevat que té un pas més al final: la distribució dels objectes; les classes i objectes que tingui el programari són els mateixos independentment que hagi d'estar distribuït o no, ja que estan determinats per la funcionalitat.

6.2.1. Ús del diagrama de desplegament

En el cas que el disseny es faci orientat a objectes i amb UML, és clar que el diagrama de desplegament es presta molt bé a descriure la distribució d'objectes entre els nodes. De fet, però, el que es distribuirà no seran objectes i classes individuals, sinó components més complexos.

6.2.2. Distribució dels diferents tipos de classes d'anàlisi

Una manera senzilla de distribuir els objectes és partir de la distinció entre classes de frontera, d'entitats i de control; podem establir aquestes regles orientatives:

a) Les classes de frontera s'han d'ubicar a les màquines en què treballen els usuaris directament.

b) Les classes d'entitats i els gestors de disc respectius estaran normalment en les màquines servidores que tenen les bases de dades.

c) Les classes de control es col·locaran justament amb les de frontera o bé amb les d'entitats segons que tinguin més interacció amb unes o altres.

6.2.3. Un paradigma alternatiu

Scott Ambler, basant-se en tècniques de Rebecca Wirfs-Brock i altres, proposa una tècnica per a la distribució dels objectes en arquitectures client/servidor de *n* capes sense agents mòbils. Distingeix cinc **tipus de classes**:

- Classes de la interfície amb l'usuari.
- Classes del negoci/domini.
- Classes de procés, que implementen processos que afecten diverses classes del domini.
- Classes de persistència, que poden ser gestors de disc o bé *frameworks*.
- Classes del sistema, com ara les de comunicació entre processos.

La **distribució dels objectes** es faria en nou passos:

1) Distribució de les altres classes abans que les del negoci/domini. Les classes de la interfície amb l'usuari s'assignarien sistemàticament als clients, les de persistència als nodes on hi ha d'haver la base de dades, si és única, i si no en un node a part. Les classes del sistema que corresponen a serveis d'ús general es replicarien a tots els nodes, mentre que les de serveis més específics* es posarien en un node a part.

*** Per exemple, seguretat.**

2) Definir el contracte (interfície pública) de les classes.

3) Simplificar les jerarquies d'especialització i agregació. Si una subclasse no té contracte propi, anirà on vagi la superclasse; un component per composició anirà amb la classe composta.

4) Identificar els components potencials del domini. Un component és un conjunt de classes que col·laboren per a oferir un conjunt de contractes que sigui coherent vist des de l'exterior. Se suposa que hi ha molta més circulació d'informació entre les classes d'un component que entre component i l'exterior. Una classe servidora amb diversos clients probablement haurà de ser un component a part, igual que aquelles classes que són únicament clients, mentre que si una classe servidora té un sol client es poden posar o bé plegades o bé en màquines diferents unides per una connexió d'alta velocitat. Dues classes que col·laborin molt sovint aniran dins el mateix component.

5) Definir els contractes pel que fa a component del domini. Si tot el contracte d'una classe servidora està inclòs dins el contracte de component al qual pertany, probablement serà millor de segregar-la com a component a part; en canvi, si cap operació del contracte d'una classe no forma part del contracte del component al qual pertany, llavors ha de constituir un subsistema intern del component.

6) Simplificar els contractes dels components, agrupant operacions amb vista a reduir el nombre de tipus de missatge diferents.

7) Assignar components a nodes mitjançant el diagrama de desplegament. Caldrà buscar la minimització del trànsit per la xarxa. La distribució s'haurà de provar, i si cal, canviar-la.

8) Afegir classes complementàries, com ara classes que fan d'interfície dels components.

9) Distribuir les classes de la interfície amb l'usuari segons els casos d'ús que tindran lloc a cada node.

Resum

S'han vist els principis i modalitats més significatius dels entorns distribuïts i, en particular, de les diferents arquitectures de maquinari i programari distribuïdes, i es culmina amb el *middleware*, dins el qual hem estudiat CORBA, que, ara com ara, és l'arquitectura dominant en aquest camp; però hem vist també RMI, que és el suport d'objectes distribuïts en Java, una tecnologia molt més senzilla que CORBA, però que, tanmateix, pot ser suficient en molts casos.

S'ha tractat també la tecnologia de documents compostos, principalment OLE/COM/DCOM. També s'han fet algunes consideracions sobre aquells aspectes del disseny de programari orientat a objectes que són específics del programari distribuït.

Exercicis d'autoavaluació

1. Expliqueu la diferència entre *entorn distribuït* i *sistema obert*.

2. En què beneficien els sistemes oberts les empreses usuàries de programari?

3. Per què es pot dir que l'arquitectura client/servidor afavoreix l'ús d'interfícies gràfiques?

4. Com s'aconsegueix més capacitat de creixement del nombre de clients a base de passar d'una arquitectura client/servidor de dues capes a una de tres?

5. Com és que a CORBA no fa falta el concepte de *classe*?

6. Quina és la diferència entre *operació* i *mètode* en CORBA?

7. Les invocacions dinàmiques fan servir esquelets dinàmics?

8. Per a què serveixen els canals d'esdeveniments?

9. Què és un objecte factoria?

10. Què fa el servei de propietats?

11. Què vol dir externalitzar un objecte?

12. Quins llenguatges de consulta suporta el servei de consultes?

13. Quina diferència hi ha entre els serveis comuns horitzontals i verticals?

14. Quina relació hi ha entre GIOP i IIOP?

15. Expliqueu la diferència entre *linking* i *embedding*.

16. Què és un contenidor?

17. Quines són les modalitats de transferència de dades en OLE/COM?

18. Què són els *monikers*?

19. Què vol dir *automatització* a COM?

20. Què és un OCX?

Solucionari

Exercicis d'autoavaluació

1. Un sistema obert és un entorn distribuït muntat a partir de components que compleixen normes determinades.

2. D'una banda, tenen menys problemes de compatibilitat entre productes; de l'altra, es beneficien d'un mercat on competeixen més productes, cosa que fa que els productes siguin millors, més barats i més variats.

3. Les interfícies gràfiques fan servir molta memòria i capacitat de procés i, per tant, generalment només seran viables si cada client suporta la seva, ja que si el servidor hagués de suportar les de tots els clients hauria de tenir unes característiques que en molts de casos ho farien inviable.

4. Quan el servidor ha arribat al nombre de clients màxim es pot afegir un segon servidor, si els dos servidors estan coordinats per un ordinador al tercer nivell.

5. Quan els clients invoquen les requestes fan referència a una interfície, i els servidors executen les requestes emprant esquelets derivats de les interfícies i objectes concrets. Per això, en CORBA no cal el concepte de *classe*.

6. A les interfícies hi ha operacions definides, a les quals els clients fan referència en les requestes; els mètodes existeixen únicament a la banda del servidor, i un mètode s'executa quan s'executa una requesta que demana l'operació que correspon al mètode. Un mètode correspon a una operació, i a una operació pot correspondre un mètode en cada servent que implementa la interfície a la qual pertany.

7. Els poden usar, però no és imprescindible. Un dels aspectes de l'aïllament que estableix l'ORB entre clients i servidors és que, d'una banda, els clients invoquen les requestes sense haver de tenir en compte si s'implementaran amb un esquelet estàtic o dinàmic, i, de l'altra, els servidors implementen les requestes de manera independent de si han estat demanades en invocació estàtica o dinàmica.

8. Serveixen perquè diversos subministradors d'esdeveniments es puguin connectar alhora a diversos consumidors.

9. Un objecte factoria serveix per a crear-ne un altre (un objecte no es pot crear a si mateix). Els fa servir el servei de cicle de vida, per a objectes de qualsevol interfície, i també altres serveis d'objectes per a crear objectes d'algunes interfícies concretes.

10. El servei de propietats permet que els objectes puguin tenir propietats dinàmiques, això és, propietats que es defineixen en temps d'execució (a diferència dels atributs, el valor dels quals generalment es pot modificar en temps d'execució, però no pas la definició de l'atribut).

11. Externalitzar un objecte vol dir convertir-lo en una cadena de caràcters, per exemple, per a enviar-lo a un altre ORB, el qual l'internalitzarà i el convertirà en un objecte en el seu format.

12. El servei de consultes suporta diverses modalitats d'SQL i d'OQL de l'OMG.

13. Els serveis horitzontals són aplicables a qualsevol aplicació, mentre que els verticals són propis d'un domini (parcel·la de la realitat) concret.

14. GIOP és un protocol general de comunicació entre ORB, i IIOP és GIOP per al cas concret en què la comunicació es fa per mitjà del protocol d'Internet, TCP/IP.

15. Amb *embedding,* el document component, inicialment independent, es copia físicament dins el document compost i es desa amb aquest, de manera que si després es fan canvis en l'original ja no es reflecteixen en el document compost. Amb *linking,* el document compost no conté el document component, sinó solament un punter a aquest, i només s'incorpora el component quan s'edita o imprimeix el document compost, i per tant sempre s'incorpora la darrera versió.

16. Un contenidor és una aplicació d'OLE que conté un document compost.

17. *Drag and drop* (arrossegar amb el ratolí), copiar i enganxar passant pel portapapers, i enllaçar.

18. Els *monikers* són objectes que són àlies permanents d'altres.

19. L'automatització consisteix en la programació i execució de *scripts*, que són petits programes lligats a un document i que s'executen quan es produeixen determinats esdeveniments en relació amb aquest.

20. Un OCX és un component d'OLE que inclou un servidor *in-process* i un servidor d'automatització.

Glossari

arquitectura client/servidor
Sistema distribuït en què un programa, el servidor, gestiona un recurs compartit en relació amb el qual altres programes i els clients poden demanar funcions determinades.

COM (*Component Object Model*)
Eina de gestió d'objectes que s'encarrega de la gestió de documents d'OLE.

CORBA (*Common Object Request Broker Architecture*)
Especificacions d'un *middleware* per a objectes distribuïts elaborat per l'OMG amb vista al fet que esdevingui un estàndard.

esquelet
Correspondència entre les operacions d'una interfície i els mètodes dels servents que la implementen. Depèn de l'ORB (és a dir, el seu format no és estàndard) i del llenguatge de programació. S'obté en compilar la interfície.

GIOP (*General InterORB Protocol*)
Protocol general per a la comunicació directa entre ORB.

IIOP (*Internet InterORB Protocol*)
Forma concreta que pren el GIOP en el cas que la connexió entre ORB es faci per mitjà del protocol d'Internet, TCP/IP.

invocació dinàmica
Invocació d'una requesta en la qual la requesta es construeix just abans d'invocar-la, en temps d'execució.

middleware
Programari que fa el paper d'intermediari entre múltiples clients i múltiples servidors.

OCX (*Custom Control*)
Combinació d'un servidor *in-process* i un d'automatització. Com a fitxer té l'extensió .ocx.

OLE (*Object Linking and Embedding*)
Eina de Microsoft per a integrar dins un document marc documents gestionats per diferents aplicacions i dades multimèdia.

ORB (*Object Request Broker*)
Pseudoobjecte que constitueix aquella part de CORBA que s'encarrega de transmetre les requestes dels objectes clients als objectes servidors.

POA (*Portable Object Adapter*)
Component de CORBA, a la banda del servidor, que s'encarrega de transmetre les requestes rebudes de l'ORB als servents que implementen els objectes servidors als quals estan adreçades.

requesta
Dins CORBA, missatge d'un objecte client a un objecte servidor per a demanar l'execució d'una operació continguda dins una interfície suportada per aquest darrer.

RMI (*Remote Method Invocation*)
Part de Java que suporta la invocació d'operacions entre objectes distribuïts.

servent
Dins CORBA, implementació d'una interfície a la qual el POA corresponent pot encomanar l'execució d'una requesta dins el servidor.

sistema obert
Sistema distribuït basat en un conjunt de normes àmpliament acceptades a nivell internacional.

Bibliografia

Orfali, R. i altres (1999). *Client/Server Survival Guide* (3a. ed.). John Wiley & Sons.

Slama, D. i altres (1999). *Enterprise CORBA*. Prentice Hall.

Rock-Evans, R. (1998). *DCOM Explained*. Digital Press.

Gerarthy, R. i altres (1999). *DCOM-CORBA Interoperability*. Prentice Hall.